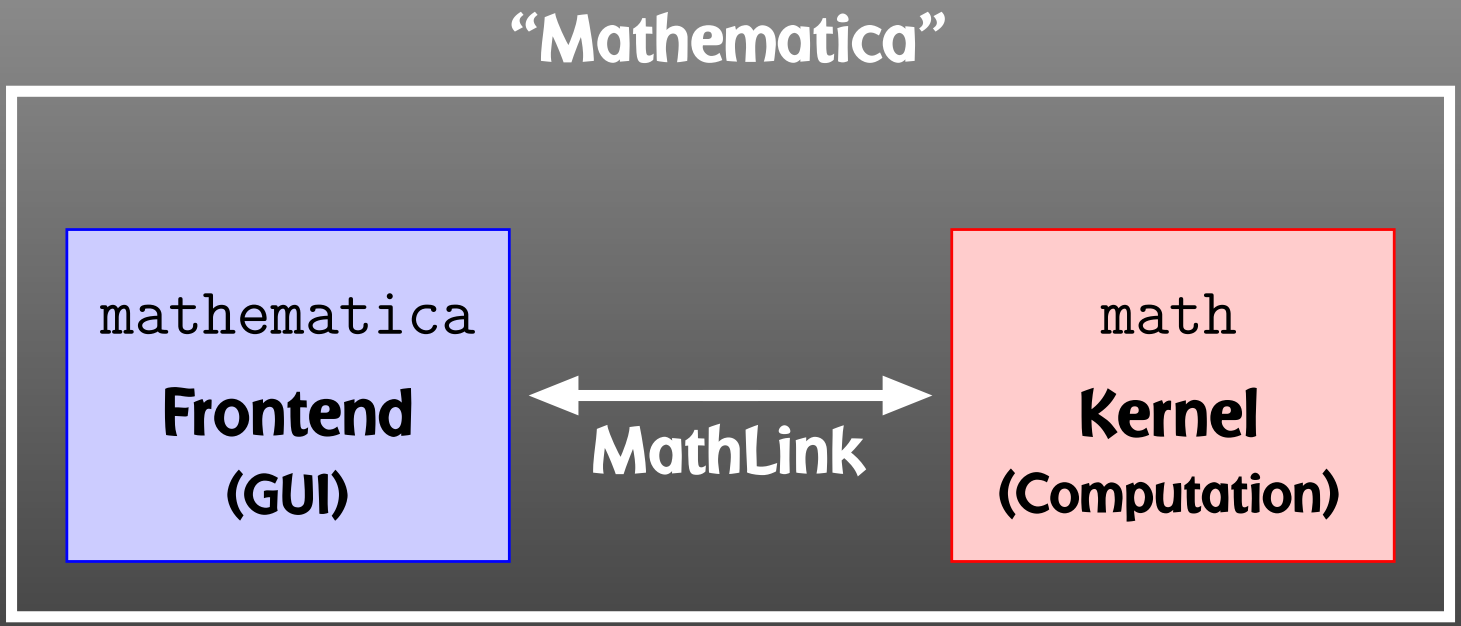
Getting most out of Mathematica

Thomas Hahn

Max-Planck-Institut für Physik
München

Mathematica Components



Why I hate the Frontend

Frontend:

- ☺ Nice formatting
- ☺ Documentation
- ☺ Ease of use
- ☹ No obvious relation between screen and definitions
- ☹ Always interactive
- ☹ Slow startup

Kernel:

- ☹ Text interface
- ☹ No pretty-printing
- ☺ 1-to-1 relation to definitions
- ☺ Interactive and non-interactive
- ☺ Scriptable
- ☺ Fast startup

Plan

- Program smart!
- Parallelize!
- Script! Distribute! Automate!
- Crunch numbers outside Mathematica!

But: don't overdo it.
If your calculation takes 5 min in total, don't waste time improving.

Program smart!

List-oriented Programming

Using Mathematica's list-oriented commands is almost always of advantage in both speed and elegance.

Consider:

```
tab = Table[Random[], {10^7}];

test1 := Block[ {sum = 0},
  Do[ sum += tab[[i]], {i, Length[tab]} ];
  sum ]

test2 := Apply[Plus, tab]
```

Here are the timings:

```
Timing[test1][[1]] ☞ 8.29 Second
Timing[test2][[1]] ☞ 1.75 Second
```

More Speed Bumps

Consider:

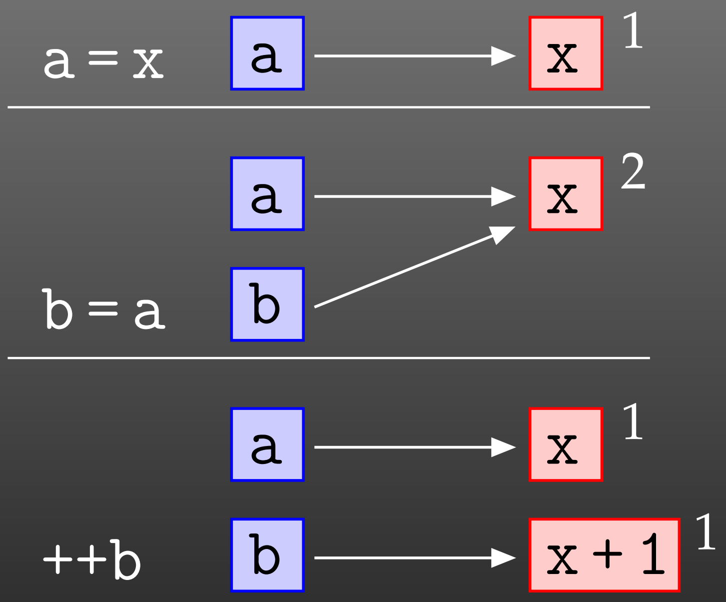
```
tab = Table[Random[], {10^5}];

test1 := Block[ {res = {}},
  Do[ AppendTo[res, tab[[i]]], {i, Length[tab]} ];
  res ]

test2 := Block[ {res = {}},
  Do[ res = {res, tab[[i]]}, {i, Length[tab]} ];
  Flatten[res] ]
```

The timings:

```
Timing[test1][[1]] ☞ 19.47 Second
Timing[test2][[1]] ☞  0.11 Second
```

Reference Count

Assignments that don't change the content make no copy but just increase the Reference Count.



Reference Count and Speed

```
test1 := ...
  ... AppendTo[res, tab[[i]]] ...
  res

test2 :=
  ... res = {res, tab[[i]]} ...
  Flatten[res]
```

`test1` **has to re-write the list every time an element is added:**

{}   {1}   {1,2}   {1,2,3}   ...

`test2` **does that only once at the end with** `Flatten`**:**

{}   {{},1}   {{{},1},2}   {{{{},1},2},3}   ...

More Programming Wisdom

- **Michael Trott**
  The Mathematica Guidebook
  for {Programming, Graphics, Numerics, Symbolics} (4 vol)
  Springer, 2004–2006.

Parallelize!

Parallel Kernels

Mathematica has built-in support for parallel Kernels:

```
LaunchKernels[];
ParallelNeeds["mypackage`"];

data = << mydata;
ParallelMap[myfunc, data];
```

Parallel Kernels count toward Sublicenses.
 # Sublicenses = 8 × # interactive Licenses.

MPP: 35 interactive licenses (5k€ each), 288 sublicenses.

Parallel Functions

- More functions:

```
ParallelArray  ParallelEvaluate  ParallelNeeds
ParallelSum    ParallelCombine   ParallelTable
ParallelDo     ParallelProduct   ParallelTry
ParallelMap    ParallelSubmit
DistributeDefinitions  DistributeContexts
```

- Automatic parallelization (so-so success):
  `Parallelize[`*expr*`]`
- 'Intrinsic' functions (e.g. `Simplify`) not parallelizable.
- Multithreaded computation partially automatic (OMP) for some numerical functions, e.g. `Eigensystem`.
- Take care of side-effects of functions.
- Usual concurrency stuff (write to same file, etc).

Script! Distribute! Automate!

Scripting Mathematica

Efficient batch processing with Mathematica:

Put everything into a script, using sh's Here documents:

```
#! /bin/sh ................. Shell Magic
math << \_EOF_ ............ start Here document (note the \)
  << FeynArts`
  << FormCalc`
  top = CreateTopologies[...];
  ...
_EOF_ ..................... end Here document
```

Everything between "<< *tag*" and "*tag*" goes to Mathematica as if it were typed from the keyboard.

Note the "\\" before *tag*, it makes the shell pass everything literally to Mathematica, without shell substitutions.

Scripting Mathematica

- Everything contained in **one compact shell script,** even if it involves several Mathematica sessions.
- Can combine with arbitrary shell programming, e.g. can use **command-line arguments** efficiently:

```
#! /bin/sh
math -run "arg1=$1" -run "arg2=$2" ... << \END
  ...
END
```

- Can easily be **run in the background,** or combined with utilities such as **make.**

Debugging hint: `-x` **flag** makes shell echo every statement,

```
#! /bin/sh -x
```

Crunch numbers outside Mathematica!

Code generation

- **Conversion** of Mathematica expression to Fortran/C **painless.**
- Optimized output can **easily run faster** than in Mathematica.
- Showstopper: Functions not available in Fortran/C, e.g. `NDSolve`, `Zeta`. Maybe 3rd-party substitute (GSL, Netlib).
- Mathematica has built-in C-code generator, e.g.

```
myfunc = Compile[{{x}}, x^2 + Sin[x^2]];
Export["myfunc.c", myfunc, "C"]
```

  But no standalone code: shared object for use with Mathematica (i.e. also needs license).

- FormCalc's code-generation functions produce optimized standalone code.

Code-generation Functions

FormCalc's code-generation functions are public and disentangled from the rest of the code. They can be used to **write out an arbitrary Mathematica expression as optimized Fortran or C code:**

- *handle* `= OpenCode["`*file.F*`"]`
  opens *file.F* as a Fortran file for writing,
- `WriteExpr[`*handle*`, {`*var*` -> `*expr*`, ...}]`
  writes out Fortran code which calculates *expr* and stores the result in *var*,
- `Close[`*handle*`]`
  closes the file again.

Code generation

Traditionally: Output in Fortran.
Code generator is meanwhile rather sophisticated, e.g.

- **Expressions too large** for Fortran are split into parts, as in

```
var = part1
var = var + part2
...
```

- **High level of optimization,** e.g. common subexpressions are pulled out and computed in temporary variables.
- **Many ancillary functions** make code generation versatile and highly automatable, such that the resulting code needs few or no changes by hand:
  `VarDecl`, `ToDoLoops`, `IndexIf`, `FileSplit`, ...

C Output

- **Output in C99** makes integration into C/C++ codes easier:

  ```
  SetLanguage["C"]
  ```

Code structured by e.g.

- **Loops and tests handled through macros,** e.g.
  `LOOP(var, 1,10,1) ... ENDLOOP(var)`
- **Sectioning by comments,** to aid **automated substitution** e.g. with `sed`, e.g. `* BEGIN VARDECL ... * END VARDECL`
- Introduced **data types** `RealType` **and** `ComplexType` for better abstraction, can e.g. be changed to different precision.

MathLink

The **MathLink API** connects Mathematica with external C/C++ programs (and vice versa). **J/Link** does the same for Java.

```
:Begin:
:Function:      copysign
:Pattern:       CopySign[x_?NumberQ, s_?NumberQ]
:Arguments:     {N[x], N[s]}
:ArgumentTypes: {Real, Real}
:ReturnType:    Real
:End:

#include "mathlink.h"

double copysign(double x, double s) {
  return (s < 0) ? -fabs(x) : fabs(x);
}

int main(int argc, char **argv) {
  return MLMain(argc, argv);
}
```

For more details see arXiv:1107.4379.